UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

SISTEMA DE BÚSQUEDA Y ANÁLISIS BASADO EN TWITTER

Daniel Garnacho Martín
Tutor: Álvaro Ortigosa Juárez

JUNIO 2015

Resumen

Las redes sociales están en constante crecimiento y cada día son más los datos que producen. Esta enorme cantidad hace necesario la introducción de análisis automáticos ya que empresas y gobiernos no pueden emplear a personas suficientes para leer cada publicación en las redes en tiempo real. En concreto, Twitter, la red social que trataremos, con cerca de trescientos millones de usuarios activos y trescientos cincuenta millones de publicaciones al día, supone un reto añadido. En este Trabajo Fin de Grado se presenta una herramienta que unifica la búsqueda en la red social y el almacenamiento en local de datos recuperados junto con técnicas de clasificación automática basada en análisis de texto. Se ha intentado simplificar el acceso a los datos con una interfaz web, junto con una capa de abstracción en el entrenamiento de los clasificadores automáticos. El TFG se divide en dos herramientas principales: el demonio de tareas que se ha creado para automatizar la recuperación y clasificación de tweets, y la herramienta web que muestra las publicaciones con un análisis sencillo, ayuda a crear patrones de entrenamiento y programar las tareas en segundo plano. Se ha buscado minimizar el tiempo de espera para mejorar la experiencia de uso del usuario, suponiendo un reto de sincronización entre el servidor y la aplicación. Por úlitmo, se ha comprobado que el clasificador más óptimo es el perceptrón multicapa, que aunque es más lento en entrenamiento, obtiene menor error y a la vez es más rápido que otros clasificadores en tiempo de explotación.

Palabras Clave

Twitter, API de Búsqueda Twitter, Interfaz Web, Clasificación Automática, Python.

Abstract

Social networks are constantly growing and every day they produce more data. This huge amount makes necessary the introduction of automatic analysis as companies and governments cannot employ enough staff to read each networks' publications in real-time. Specifically, Twitter, the social network that we discuss, with about three hundred million active users and three hundred fifty million posts per day, represents an additional challenge. In this Final Project we present a tool that unifies searching the social network and local storage of data retrieved with automatic classification techniques based on text analysis. We attempted to simplify the access to the data with a web interface, together with a layer of abstraction in training automatic classifiers. The TFG is divided into two main tools: the demon of tasks that has been created to automate the retrieval and classification of tweets, and the web tool that shows publications with a simple analysis, helps in creating training patterns and schedule tasks in the background. We tried to minimize the waiting time to improve the user's experience, confronting the challenge of synchronization between the server and the application. Finally, it was found that the optimal classifier is the Multilayer Perceptron, although it is slower in training, it gets less error and is faster than other classifiers while operating.

Index Terms

Twitter, Twitter Search API, Web Interface, Automated Classification, Python.

Índice

1. Introducción

1.1.Motivación

Con el creciente interés y uso de las redes sociales es una necesidad real el análisis de los datos que estas redes nos proporcionan, en concreto, en Twitter se publican 15.000 tweets por segundo en todo el mundo, por lo que se hace necesario que el análisis sea lo más automático posible y a la vez simplificar la tarea a los usuarios. Bajo estas premisas se ideó este trabajo fin de grado, para ello se ha creado una herramienta de recuperación de datos de la red social, aparte de la recuperación se ha creado un módulo de clasificación de texto optimizado para la poca información que proporciona la red social, a la vez la herramienta proporciona análisis de la frecuencia de las publicaciones.

No existen en el mercado ninguna herramienta que unifique la búsqueda y recuperación de datos con la clasificación automática para facilitar la tarea de análisis de la red social, se ha decidido crear un herramienta web y así poder llegar potencialmente a más usuarios.

1.2.Objetivos

Los objetivos de este trabajo fin de grado son:

- Recuperación información desde la red social. Utilización de la API de Twitter, conocer el límite de consultas y velocidad en la recuperación de publicaciones.
- Saber que se va a guardar y de qué manera. Evitar publicaciones duplicadas, filtrar la información relevante, actualizarla.
- Entrenamiento y Clasificación de tweets. Conocer los mejores algoritmos para realizar la clasificación de texto de 140 caracteres máximo que utiliza la red social.
- Interfaz web sencilla. No todos los usuarios de la aplicación han de conocer cómo funciona la clasificación automática, pero si han de ser capaces de entrenar un clasificador desde la aplicación y poder usarlo sin conocimiento de herramientas externas.

1.3.Estructura del documento

En la sección 2 se realizará una exposición del estado del arte, con la finalidad de conocer las herramientas actuales en torno a la red social que nos ocupa en este trabajo, las librerías de clasificación automáticas que se podrían haber utilizado, así como una pequeña comparativa de servidores web en concreto los que se ejecutan sobre Python.

En la sección 3 se enunciarán los requisitos funcionales, un diseño general de los módulos y como se comunican entre ellos, por último en esta sección se verá el diseño en detalle de los módulos.

En la sección 4 se ha querido añadir una pequeña guía sobre la instalación de la aplicación, dependencias que existen tanto Python, JavaScript y programas externos, también se han querido dar unas pautas para la instalación del Daemon creado.

En la sección 5 se enunciarán las pruebas unitarias más importantes, de integración y de la API de Twitter.

En la sección 6 se incluyen las conclusiones del proyecto software y posibles ideas como trabajo futuro, este trabajo puede dar pie a muchos otros proyectos.

1.4.Notación y terminología

Twitter: Red social en la que las publicaciones son como máximo 140 caracteres.

Tweet: Publicación o actualización en el estado de un usuario en la red social Twitter.

Machine Learning: aprendizaje automático.

Perceptrón multicapa: clasificador que utiliza una red neuronal artificial y utiliza retropropagación como algoritmo de aprendizaje.

Demonio: Proceso en segundo plano, programa residente.

API: Interfaz de Programación de Aplicaciones *(Application Programming Interface)*

2. Estado del arte

Las redes sociales en informática suponen un reto por la cantidad de información que contienen y a la vez se necesitan de herramientas útiles con las que analizarlas. En esta sección se enunciarán las soluciones propuestas actualmente que han resultado como inspiración a la realización de este trabajo fin de grado.

2.1. Análisis de Twitter

2.1.1. Twitter Analitics (1)

Twitter Analytics es una herramienta proporcionada por la red social para mejorar la influencia y conocer más datos del usuario de la red social, solo da información del usuario registrado y no es capaz de ofrecer análisis de otros usuarios. Ofrece una sección de seguidores, en las que busca intereses y zona geográfica de los seguidores de tu cuenta.

2.1.2. Topsy (2)

Topsy es una herramienta que recopila todos los tweets de la red social y genera análisis de frecuencias, es capaz de ordenar los resultados por el tiempo de creación y por relevancia. Es muy interesante porque recupera todos los tweets desde la creación de la red social, no como la API de twitter, que no te asegura el acceso a los datos. No realiza análisis de los seguidores.

2.1.3. Twitonomy (3)

Twitonomy es quizás la herramienta más conocida y ofrece herramientas similares que Twitter Analytics pero para todos los usuarios de la red.

La mayoría de las herramientas están centradas en el marketing en las redes sociales y no proporcionan los datos para que se puedan analizar por software externo, también obligan a dar permisos que comprometen la privacidad del usuario que las usa, no se ha encontrado ninguna herramienta que realice filtrado por tweets como la herramienta desarrollada.

2.2.Herramientas de Aprendizaje Automático

2.2.1. Weka (4)

Weka es una aplicación y librería de aprendizaje automático, desarrollada en Java lo que permite ser ejecutado en los sistemas operativos más comunes. Su utilización es sencilla gracias a la interfaz, también se proporciona ejecución por línea de comandos, pudiendo guardar el modelo, no se ha optado por su uso en el proyecto dado que se quería mantener la independencia de librerías y otros sistemas externos al mínimo así como la clasificación de elemento a elemento podría haber resultado un problema de integración con Weka.

2.2.2. Scikit-learn (5)

Scikit-learn al igual que Weka es una librería de aprendizaje automático, desarrollada sobre librerías matemáticas en Python, quizás la mejor opción si se desea hacer clasificación con este lenguaje, no implementa el perceptrón multicapa que es el algoritmo que mejor resultado ha dado en las pruebas, también es uno de los algoritmos con menor tiempo de clasificación.

2.2.3. Apache Mahout (6)

Apache Mahout puede realizar tareas de clasificación y demás, la diferencia con los sistemas anteriores es que puede funcionar en varias máquinas de manera simultánea, utilizando Spark, muy interesante si se desea aumentar en escala el procesamiento de datos. En contra podemos decir que no ofrece muchos sistemas de clasificación.

2.3.Servidores Web

2.3.1. Apache (7)

Apache quizás sea el servidor más extendido, suele estar centrado en el desarrollo con PHP, nosotros lo usaremos como un complemento para mejorar el funcionamiento multiusuario de nuestra aplicación es fase de producción. Para poder ejecutar Python se utilizará mod_wsgi (8).

2.3.2. Django (9)

Django es un servidor Python quizás el más utilizado, se suele utilizar en proyectos software de una mayor envergadura que el diseño planteado inicialmente para nuestra aplicación por eso no se optó por su utilización. La utilización del Framework de Django dificulta la flexibilidad del proyecto pero mejora el mantenimiento.

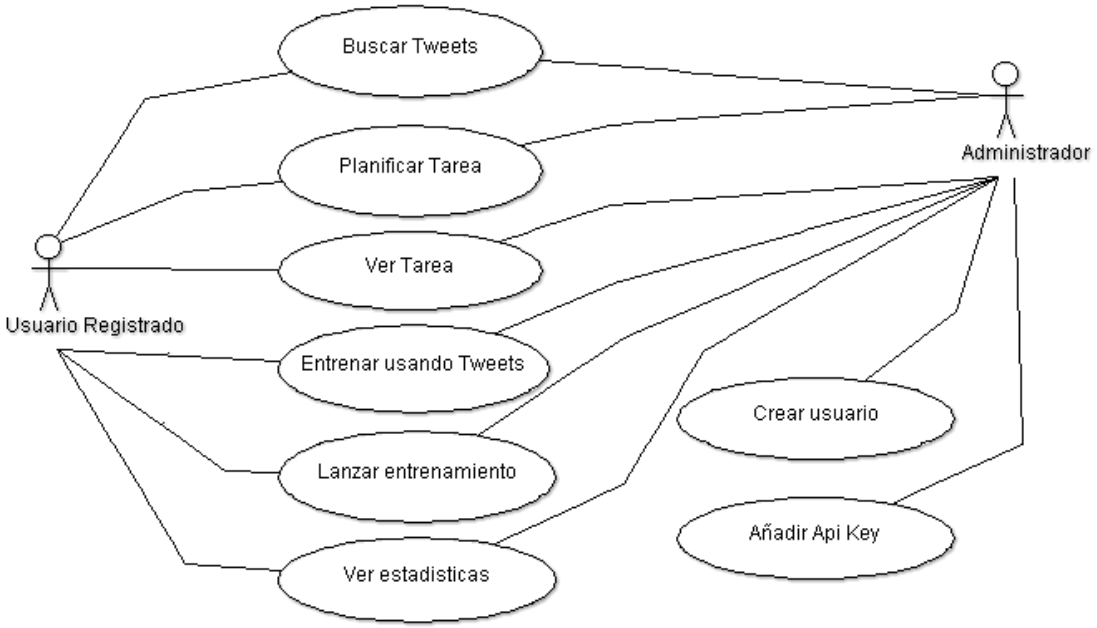
2.3.3. Flask (10)

El servidor Python por el que hemos optado es Flask, Flask nos ofrece un Framework mucho más flexible que Django, permite al programador decidir cómo se organiza la aplicación, la documentación nos proporciona ejemplos para comenzar a desarrollar rápidamente y los templates dan al programador la opción de desarrollar páginas al estilo AngularJS.

3. Diseño y desarrollo

3.1.Análisis de requisitos

En esta sección se enunciarán los requisitos funcionales y no funcionales del sistema, en la siguiente imagen podemos ver el diagrama de casos de usos obviando que se ha iniciado sesión en la aplicación.



3.1.1. Requisitos funcionales

RF.1: Acceso a la aplicación:

- **RF.1.1:** Solo un usuario registrado tiene acceso a la aplicación y debe conocer nombre de usuario y contraseña.
- **RF.1.2:** Los administradores son los únicos capaces de dar de alta nuevos usuarios en la aplicación.
- **RF.1.3:** Todos los usuarios han de acceder a la aplicación desde la página principal, el sistema es el encargado de diferenciar entre administradores o usuarios sin privilegios.

RF.2: Búsquedas:

- **RF.2.1:** Un usuario ha de ser capaz de realizar búsquedas dados uno parámetro de entrada, esta búsqueda devolverá tweets de la red social Twitter. Estos tweets pueden estar almacenados en la base de datos para un acceso más rápido pero también deberá recopilar nuevos datos.
- **RF.2.2:** Se podrán realizar dos tipos de búsqueda, búsqueda de tweets de un usuario o por contenido.
- **RF.2.3:** El contenido se mostrará de mayor a menor fecha de publicación.

RF.3: Planificación de tareas:

- **RF.3.1:** Un usuario podrá crear tareas de búsqueda, estas tareas tendrán como finalidad recopilar tweets y analizarlos (si el usuario así lo desea) durante un tiempo definido, las búsquedas son del mismo tipo que en el apartado anterior.

RF.3.2: Una tarea puede estar activa durante una semana, dos semanas o un mes.

RF.3.3: Las tareas podrán ser observadas aunque no hayan finalizado, ofrecerán los tweets recuperados y un análisis de frecuencia de los tweets recuperados.

RF.3.4: Si la tarea contiene análisis de la información recopilada, el sistema ha de ser capaz de filtrar los tweets dado el entrenamiento que haya realizado el usuario.

RF.4: Entrenamiento usando tweets:

RF.4.1: Un usuario ha de ser capaz de crear y eliminar listas de entrenamiento.

RF.4.2: Una vez creadas las listas de entrenamiento, el usuario ha de ser capaz de clasificar tweets en dos categorías, relevante o no relevante, los tweets entrenados se insertarán en la lista indicada por el usuario.

RF.4.3: Se podrá ver el contenido de las listas de entrenamiento, siendo posible cambiar el voto en cualquier momento.

RF.4.4: El usuario es el encargado de lanzar el entrenamiento una vez se haya rellenado la lista.

RF.5: Panel de administración, solo accesible por los administradores:

RF.5.1: El usuario administrador podrá dar de alta usuarios.

RF.5.2: También será capaz de añadir nuevas claves a la API de Twitter.

3.1.2. Requisitos no funcionales

RNF.1: Dado que no se conocen las competencias en tecnología del usuario final, el sistema ha de ser intuitivo utilizando patrones de diseño web comunes, pudiendo reducir el tiempo de aprendizaje a 20 minutos máximo.

RNF.2: El tiempo de carga de las páginas no debe superar el segundo y en caso de que no sea posible, se debe proporcionar información al usuario para que pueda trabajar mientras se cargan los datos solicitados.

RNF.3: El sistema debe soportar la concurrencia de al menos 10 usuarios.

RNF.4: El sistema deberá funcionar en Firefox y Google Chrome, en ningún momento se asegurará el funcionamiento en Internet Explorer.

RNF.5: El sistema debe ser capaz de almacenar al menos un millón de tweets con sus usuarios y soportar la carga de al menos 50 tareas programadas simultáneas.

RNF.6: El sistema debe tolerar fallos en la entrada de los usuarios, así como evitar inyecciones SQL

RNF.7: Si y solo si es un usuario registrado, podrá acceder al contenido de la aplicación, necesitando un gran esfuerzo de seguridad.

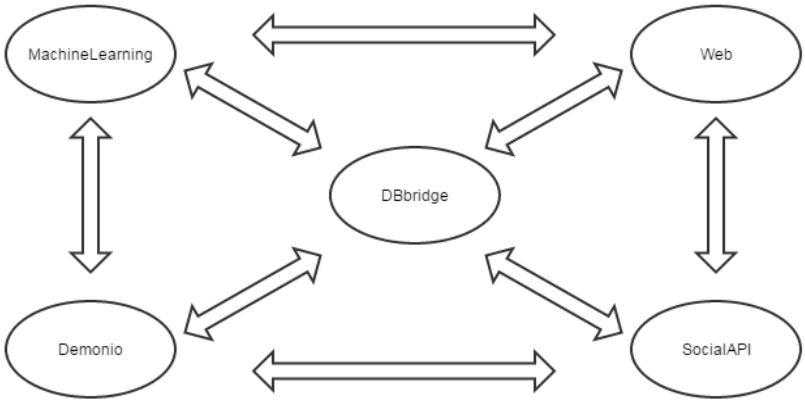
RNF.8: Si y solo si es un usuario administrador, podrá crear usuarios y añadir claves de la API de Twitter, con la seguridad que conlleva.

RNF.9: Las contraseñas en ningún momento se almacenarán en plano.

RNF.10: Si el cliente así lo solicitara debería ser sencillo incluir cifrado en las conexiones.

3.2.Diseño general

La aplicación consta de cinco módulos bien diferenciados. En primer lugar, DBbridge nos proporcionará las conexiones, escritura y lectura de datos con la base de datos PostgreSQL, MachineLearning proporciona todos los elementos necesarios para hacer clasificación automática de Tweets, Web como su nombre indica es la creación de la interfaz gráfica usando Flask como servidor, SocialAPI será la encargada de mantener la conexión con la API de Twitter, por último Demonio va a contener la funcionalidad necesaria para realizar tareas en segundo plano de recolección y análisis.



Como se puede ver, DBbridge es un módulo central por el que pasa la información que va a ser permanente todos los demás hacen uso de él. Por un lado la interfaz web es capaz de lanzar búsquedas con SocialAPI y entrenar el clasificador con MachineLearning. Demonio crea las tareas de búsqueda usando SocialAPI como base y es capaz de clasificar tweets usando MachineLearning, ningún módulo es capaz de lanzar el Demonio, si no que las tareas se programan en la base de datos y este es el encargado de leerlas. Se podría ver el diseño como un patrón MVC siendo el modelo toda la aplicación menos el módulo web, la vista las clases Python de Web y el controlador el JavaScript junto con el servidor.

3.3.Diseño detallado

En esta sección se profundizará en el diseño interno de cada uno de los módulos.

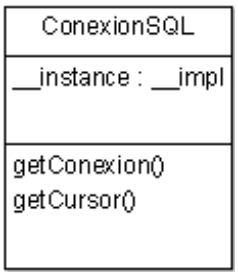
3.3.1. DBbrige

Como ya se ha mencionado anteriormente DBbridge contendrá toda la funcionalidad centrada en el uso de la base de datos, como base de datos se ha usado PostgreSQL, uno de los motores más potente, posee gran escalabilidad, Implementa el uso de rollback's, subconsultas, transacciones y claves foráneas.

Las clases que componen DBbridge son las siguientes:

ConexionSQL

ConexionSQL proporciona a todos los elementos una conexión unificada a la base de datos usando Psycopg2, implementa un patrón singleton para evitar generar conexiones innecesarias con la base de datos, este elemento es importante por si se va a utilizar la base de datos en un servidor Apache y no como un ejecutable Python.



CreaTablas

CreaTablas como su propio nombre indica, genera todas las tablas que va a hacer uso la aplicación, ejecutable Python. Las tablas junto con su contenido son las siguientes:

Users:

Información de usuarios recolectados de la red social.

- Id: identificador interno de la aplicación para el usuario, clave primaria
- id_twitter: identificador retornado por la API de Twitter
- name: nombre que retorna la API
- screen_name: nombre de usuario
- followers: número de seguidores
- location: localización
- created_at: fecha de creación de la cuenta
- last_tweet_collected: identificador del ultimo tweet recolectado del usuario

Tweets:

Información de los tweets recolectados de la red social. Contiene un índice por id_twitter y por tuser.

- Id: identificador interno de la aplicación para el tweet, clave primaria
- id_twitter: identificador retornado por la API
- status: cadena escrita en el tweet
- tuser: referencia al usuario anteriormente descrito, su id
- created_at: fecha de creación del tweet
- lang: idioma del tweet
- is_retweet: si forma parte de un retweet
- orig_tweet: si forma parte de un retweet, de que tweet viene
- favorite_count: número de favoritos
- retweet_count: número de retweets
- media_url: guarda los links de las imágenes si contiene

Twitter_tokens:

Mantiene las claves de acceso a la API de Twitter.

- Id: identificador, clave primaria
- api_key: proporcionado por Twitter
- api_key_secret: proporcionado por Twitter
- access_token: proporcionado por Twitter
- access_token_secret: proporcionado por Twitter

Tokens_count:

Lleva la cuenta del uso de cada twitter_token, para evitar llegar al límite de uso de la API Key, lo que puede conllevar al Ban de la IP, se genera una cola de prioridad.

- Id: identificador, clave primaria
- Tiempo: tiempo en el que se ejecuta la consulta
- id_token: identificador del token usado, referencia twitter_tokens (id)
- simulado: necesario para generar la cola de prioridad si no hay consultas en 15 minutos

App_users:

Usuarios de la aplicación web

- id: identificador, clave primaria
- mail: correo del usuario
- name: nombre
- institution: institución, por ejemplo UAM
- role: administrador o normal
- username: nombre de usuario, para entrar en la aplicación
- pasw: contraseña cifrada en sha256

App_searches

Mantiene las búsquedas que realiza cada usuario

- id: identificador, clave primaria
- search_string: cadena de búsqueda
- id_user: usuario que ha realizado la búsqueda, referencia app_users(id)
- number_new_tweets: número de tweets nuevos
- number_recalled_tweets: número de tweets ya existentes en la aplicación
- search_time: tiempo que ha tardado la búsqueda

Join_search_tweet:

Unión entre una búsqueda y los tweets recolectados, los dos elementos conforman la clave primaria.

- id_search: id de la búsqueda, referencia app_searches(id)
- id_tweet: id del tweet, referencia tweets(id)

Tareas_programadas:

Las tareas programadas como indica su nombre van a ser tareas que se van a poder crear desde la aplicación web y que se ejecutarán en segundo plano, buscando y analizando tweets.

- Id: identificador de la tarea
- Tipo: tipo de tarea, por ahora hay 4 tipos
- id_search: identificador de búsqueda, referencia app_searches(id)
- tiempo_inicio: tiempo en el que se programa la tarea
- tiempo_fin: cuando dejará de ejecutarse
- id_lista_entrenamiento: si es una tarea de análisis, que red neuronal tiene que usar

Listas_entrenamiento:

La aplicación es capaz de usar muchas listas de entrenamiento, para diferentes temas

- id: identificador, clave primaria
- nombre: nombre de la lista, pequeña descripción del contenido

Tweets_entrenamiento:

En esta tabla se van a guardar los tweets clasificados por el usuario.

- Id: identificador, clave primaria
- id_tweet: referencia tweets(id)
- id_lista: listas_entrenamiento(id)
- fecha_creacion: fecha en la que se añade el tweet para entrenar
- clase: clase de entrenamiento

Entrenamientos:

Contiene la información de los entrenamientos realizados.

- Id: identificador, clave primaria
- Fecha: fecha de creación del entrenamiento
- Tipo: por ahora solo hay un tipo, tweet, analiza los tweets dadas sus palabras
- fichero_arff: fichero .arff generado para entrenamiento
- fichero_json: modelo generado del entrenamiento
- porcentaje_fallo: porcentaje de fallo de entrenamiento
- id_lista_entrenamiento: referencia listas_entrenamiento(id)

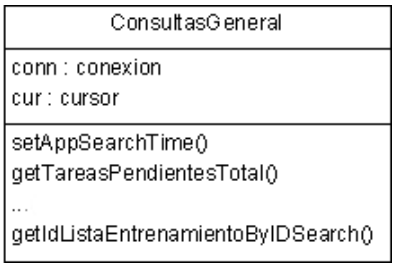
ClasificacionTweets:

Guarda un tweet clasificado

- id_tweet: identificador
- clase: clase predicha

ConsultasGeneral

La clase ConsultasGeneral proporciona una interfaz sencilla de comunicación de los componentes con la base de datos, dados unos parámetros de entrada a cada método retorna los datos extraídos, está formado por veinticinco métodos de tipo “select”, “set” y “delete”, utiliza la clase ConexionSQL.

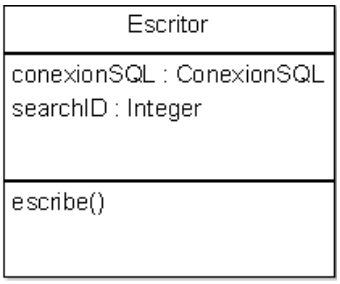


ConsultasWeb

La clase ConsultasWeb tiene la misma finalidad y extiende de ConsultasGeneral, sus consultas están más centradas a la interfaz web.

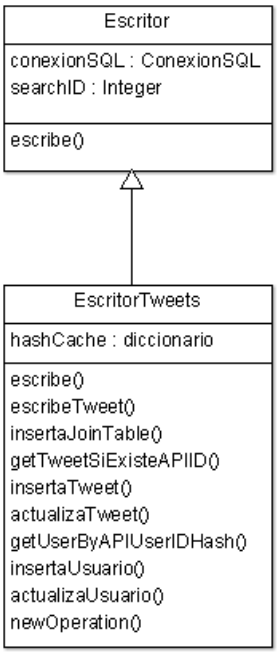
Escritor

Tiene las funciones de interfaz, para que los objetos que la implementan sean capaces de escribir la información proporcionada por una red social, se busca poder extender la aplicación de una manera sencilla, por ejemplo con más información de Twitter o con otras redes sociales.



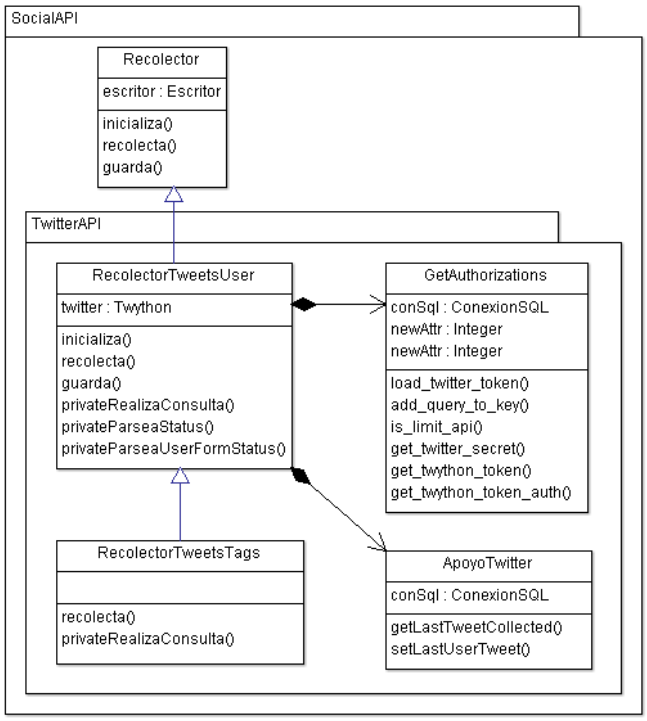
EscritorTweets

La clase EscritorTweets extiende de Escritor y se va a encargar de guardar en la base de datos un array de tweets, manteniendo las relaciones entre las tablas, recibe una lista de diccionarios, cada diccionario contiene la información del tweet preprocesada.



3.3.2. SocialAPI

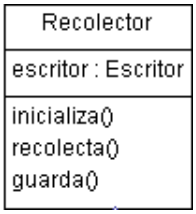
El módulo SocialAPI es el encargado de mantener las clases que van a realizar las conexiones y búsquedas con las redes sociales, en este caso sólo existe una red social, pero se ha pensado en el diseño para incluir más redes sociales, como puede ser Facebook.



Las clases explicadas en detalle, que componen SocialAPI son:

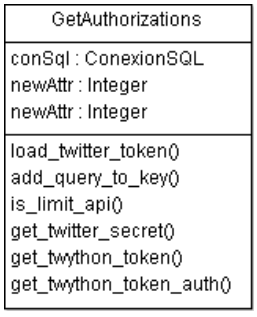
Recolector

La clase Recolector trata de unificar la funcionalidad de todos los recolectores de las redes sociales que se quieran utilizar, por lo que es una clase genérica sin funcionalidad más allá de la inicialización, esta inicialización recibe objeto de tipo Escritor, descrito anteriormente, por lo que es el encargado el recolector de llamar al escritor.



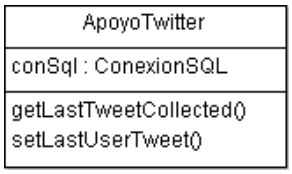
GetAuthorizations

GetAuthorizations es la encargada de proporcionar las claves de acceso a la red social Twitter, también se encarga de escribir cada vez que se realiza una consulta externa en la base de datos para evitar llegar al límite de accesos a la API.



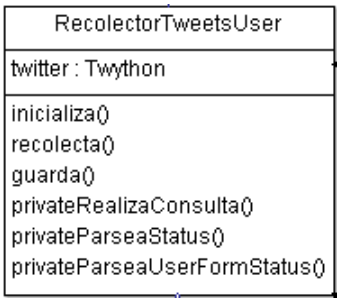
ApoyoTwitter

ApoyoTwitter hace uso de la base de datos para proporcionar información a los recolectores de datos. Trata de separar totalmente al recolector de la base de datos.



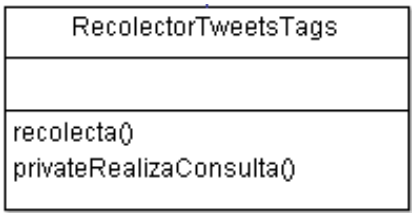
RecolectorTweetsUser

Dado un usuario, RecolectorTweetsUser va a realizar una consulta a la API de twitter, usando la interfaz de Twython (11), aparte de realizar la consulta, también va a organizar los datos para que el objeto Escritor no tenga que realizar ninguna tarea extra a su funcionalidad, extiende de Recolector.



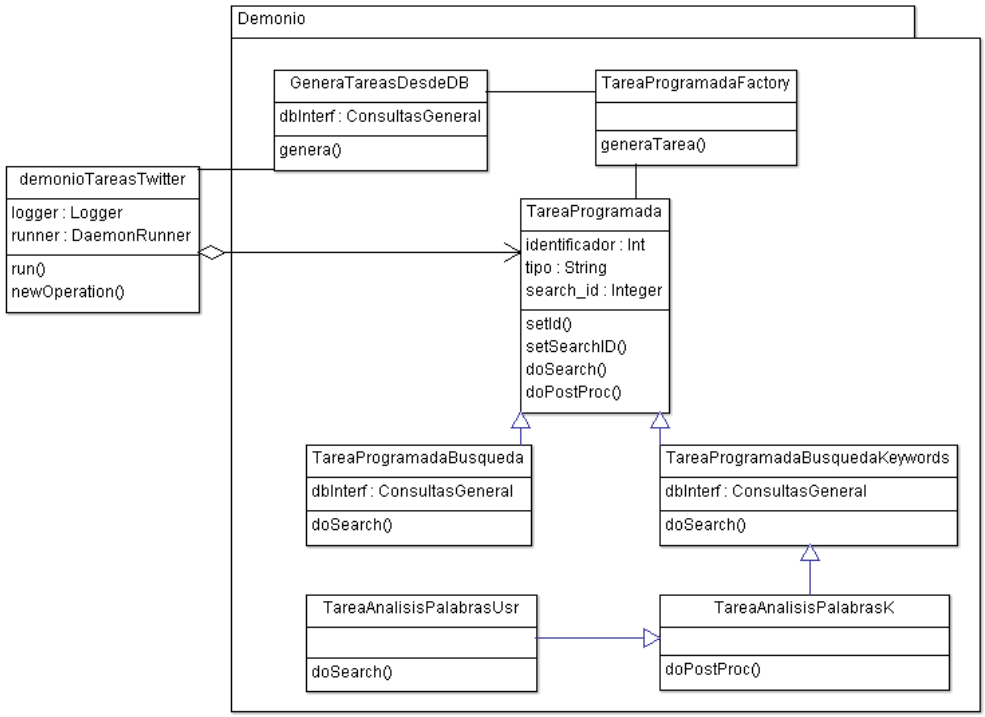
RecolectorTweetsTags

RecolectorTweetsTags extiende de RecolectorTweetsUser y al igual que esta clase, realiza una consulta a Twitter, con la particularidad de que va a pedir resultados que se ajusten a la consulta y no a un solo usuario, en el siguiente diagrama UML solo se muestran las clases que sobrescribe.



3.3.3. Demonio

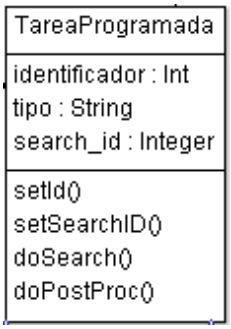
El módulo Demonio es el encargado de mantener la funcionalidad de las tareas en segundo plano, para poder realizar búsquedas y análisis durante el tiempo que esté activo el servidor y sin que el usuario deba lanzarlas continuamente, se ha optado por realizar un proceso en segundo plano que arranca con el sistema operativo, este proceso lee la base de datos y ejecuta las tareas no terminadas.



Las clases explicadas en detalle, que componen SocialAPI son:

TareaProgramada

TareaProgramada es una clase abstracta que va a unificar la funcionalidad de las tareas, tiene dos métodos principales, doSearch, realizará la búsqueda y doPostProc se va a encargar del análisis, por ahora solo se ha programado una clasificación de los tweets dadas sus palabras.



TareaProgramadaBusqueda

TareaProgramadaBusqueda extiende de TareaProgramada, se encarga de las búsquedas en segundo plano de usuario, esto es, recolectar los tweets de un usuario durante un tiempo concreto.

TareaProgramadaBusquedaKeywords

TareaProgramadaBusquedaKeywords extiende de TareaProgramada, al igual que TareaProgramadaBusqueda realiza una búsqueda a la API de Twitter, pero la búsqueda no es por usuario si no por una serie de palabras.

TareaAnalisisPalabrasK

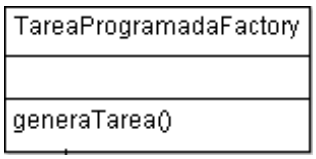
TareaAnalisisPalabrasK extiende de TareaProgramadaBusquedaKeywords, una vez que se han bajado los tweets se utiliza el módulo de MachineLearning para realizar una clasificación de los tweets recuperados, aunque se explicará a continuación los tweets se clasifican entre relevante y no relevante dada una red neuronal entrenada con tweets clasificados por el usuario.

TareaAnalisisPalabrasUsr

TareaAnalisisPalabrasUsr extiende de TareaAnalisisPalabrasK, utiliza el mismo analizador, pero la búsqueda es la misma a la de la clase TareaProgramadaBusqueda.

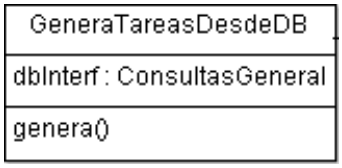
TareaProgramadaFactory

TareaProgramadaFactory implementa el patrón de diseño Factory Method, genera las tareas vistas anteriormente dado un identificador de la clase.



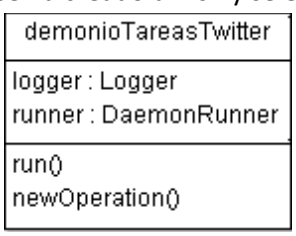
GeneraTareasDesdeDB

La clase GeneraTareasDesdeDB realiza una consulta de las tareas pendientes a la base de datos utilizando ConsultasGeneral, llama a TareaProgramadaFactory e inicializa las tareas con su identificador y el identificador de la búsqueda, devuelve una lista de tareas que va a usar la clase DemonioTareasTwitter.



DemonioTareasTwitter

DemonioTareasTwitter es la clase principal del demonio, se encarga de obtener las tareas de la base de datos y de mantener activo el bucle de tareas, cada cinco minutos lanza las tareas, primero ejecutando las búsquedas y a continuación si la tarea es de análisis ejecuta la clasificación, utiliza el módulo de Python runner.Daemon, también genera un log del sistema Linux donde escribe el número de tareas activas en cada iteración, para que el demonio arranque con el sistema operativo se ha creado un .sh y se explicará cómo usarlo.

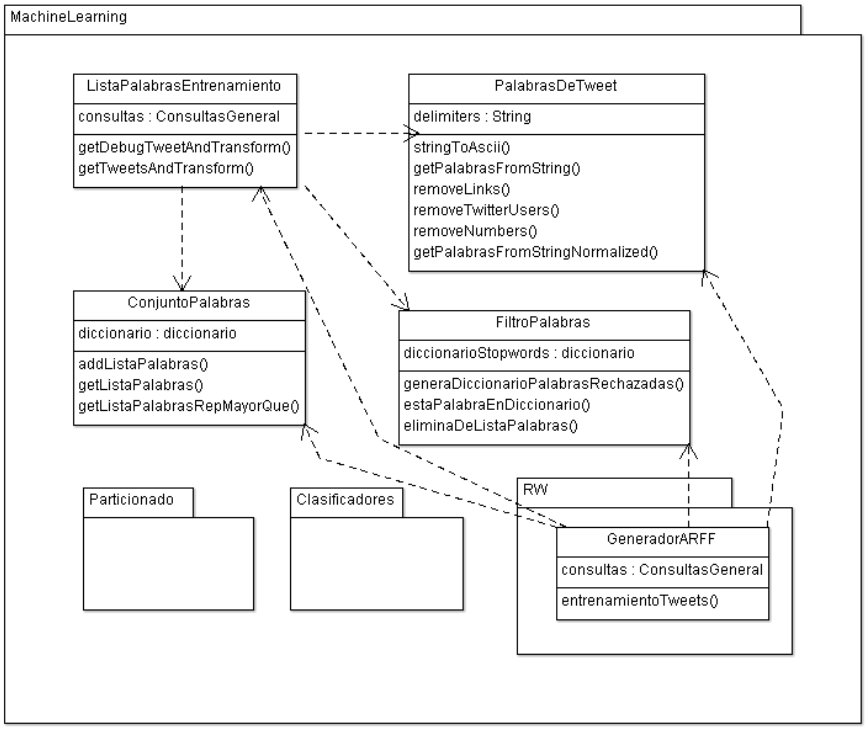


3.3.4. MachineLearning

El módulo MachineLearning como su propio nombre indica es el encargado de proporcionar la funcionalidad de aprendizaje automático, la clasificación de texto, se va a encargar de pre-procesar los datos para que se puedan usar en el entrenamiento, lanzar el entrenamiento del clasificador y dados ejemplos nuevos, clasificarlos. Este módulo es crítico por lo que ha sido probado más a fondo, la mayoría de las clases contienen pruebas unitarias y de integración.

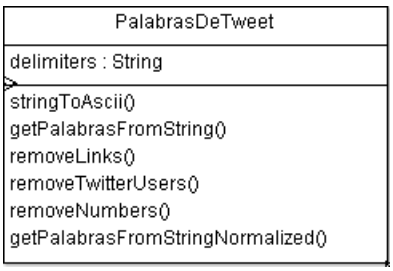
En primer momento diferenciaremos las clases que se encargan de producir la entrada que los clasificadores usan, de las clases de aprendizaje automático genérico.

Tokenización, Stopwords, Normalización.



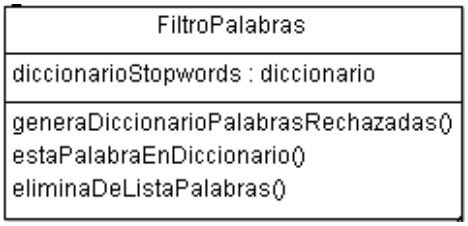
PalabrasDeTweet

PalabrasDeTweet es la clase encargada de la tokenización y normalización de texto, todos los métodos reciben un String y el método **getPalabrasFromString** es el encargado de devolver un Array de Strings, cada elemento de este Array es un token del texto, pueden estar repetidos, esta clase al ser independiente contiene pruebas unitarias y por encima se han realizado pruebas de integración.



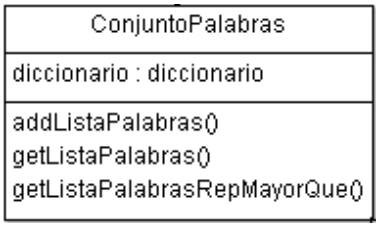
FiltroPalabras

FiltroPalabras es la clase encargada de eliminar las palabras no relevantes en español esto ayudará al aprendizaje de los clasificadores, si se desea usar más idiomas simplemente hay que añadir palabras no relevantes de ese idioma, esta clase contiene pruebas unitarias.



ConjuntoPalabras

ConjuntoPalabras es la clase encargada de mantener los tokens sin duplicar contando el número de repeticiones, el método **getListaPalabrasRepMayorQue()** nos proporcionará los tokens con un número de repeticiones mayores a uno especificado, se ha visto que resulta útil si en número de tweets es lo suficientemente grande, reduciendo así la dimensión de la entrada al clasificador.



ListaPalabrasEntrenamiento

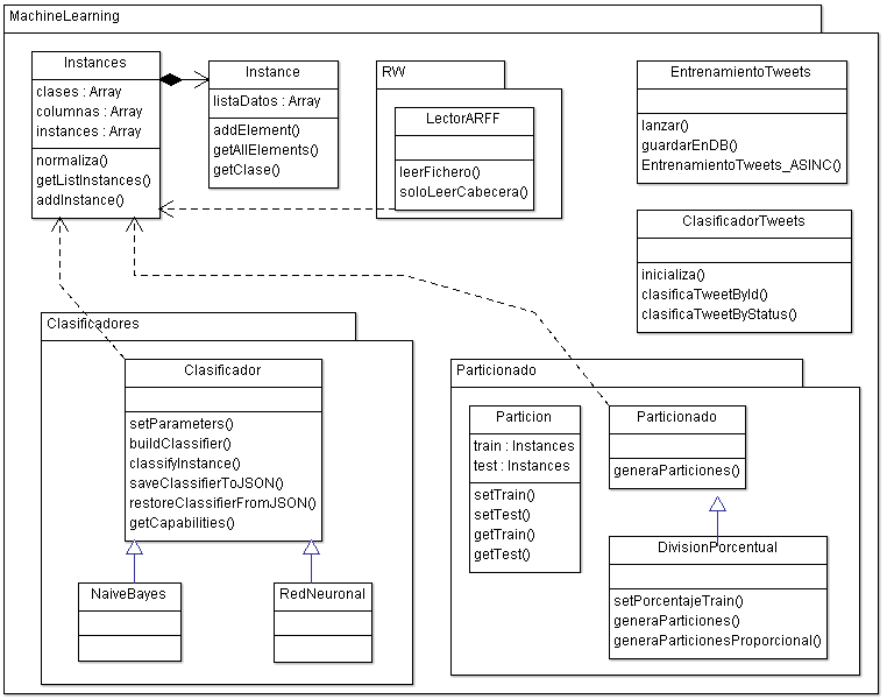
ListaPalabrasEntrenamiento esta clase se encarga de producir una bolsa de palabras a partir de una serie de tweets, utilizando las clases anteriormente mencionadas, contiene pruebas unitarias.

GeneradorARFF

GeneradorARFF es la clase encargada de generar un fichero .arff con los patrones de entrenamiento y la clasificación definida por el usuario de la aplicación, estos patrones contienen la frecuencia de cada palabra en un tweet. Los ficheros arff son usados por Weka, una librería de Machine Learning muy conocida, usar este tipo de ficheros nos ayudaría en una iteración posterior del proyecto a usar Weka en vez de la implementación propia, esta clase genera un patrón por cada tweet de entrenamiento, haciendo uso de las clases anteriores, esta clase contiene pruebas unitarias y de integración.

Aprendizaje Automático

A continuación se enunciarán las clases y métodos implementados, relacionados con el aprendizaje automático, se ha usado Weka como inspiración para el diseño.

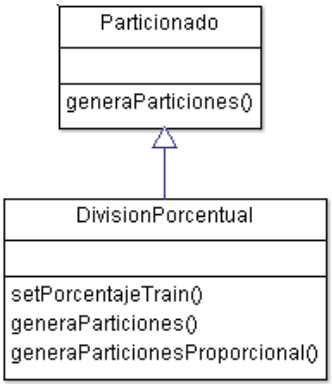


Instance

Instance es la clase principal de información para el entrenamiento de un clasificador, se acumulan en la clase Instances. Mantiene la información leída por LectorARFF, en la última posición del Array de datos se guarda la clase.

Particionado

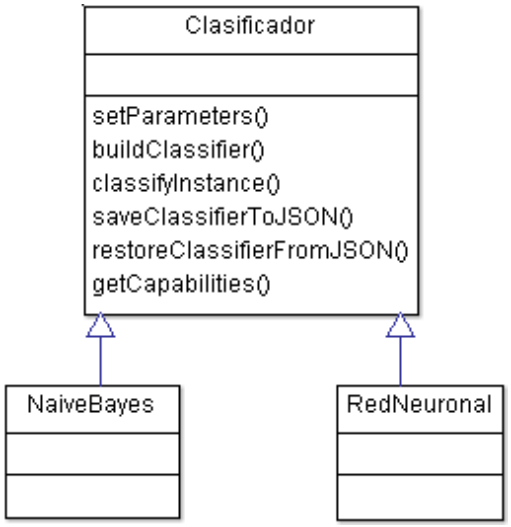
Particionado es una clase abstracta, las clases que extiendan de Particionado han de generar particiones, esto es, dividir las instancias en entrenamiento y en test, solo se ha creado un tipo de particionado, el particionado porcentual divide las instancias aleatoriamente y con una probabilidad p va a ser insertando en el conjunto de entrenamiento y con una probabilidad 1-p en el conjunto de test, la división porcentual proporcional asegura que el porcentaje de cada clase en los conjuntos nuevos va a ser proporcional a la del conjunto total de los datos.



Clasificador

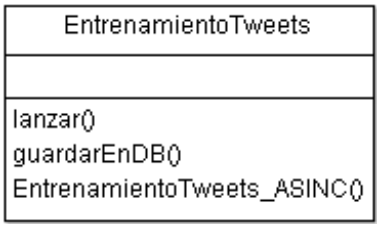
Clasificador es una clase abstracta que unifica la funcionalidad de todos los clasificadores de la aplicación, como muchos clasificadores necesitan parámetros de configuración, se ha creado un método para introducirlos de manera genérica.

Se han implementado dos clasificadores, **Naïve Bayes** suponiendo que los datos siguen una distribución normal, también se ha implementado una red neuronal usando el **perceptrón multicapa** (12) y como algoritmo de aprendizaje la retropropagación del error, este clasificador es el que mejores resultados ha dado en diferentes test, por lo que es el que se usará siempre, como las redes neuronales tienen un algoritmo de aprendizaje costoso se guardan los pesos de la red una vez aprendida para poder clasificar rápidamente los tweets según llegan, todos los clasificadores se han validado usando Weka y se han realizado pruebas de integración.



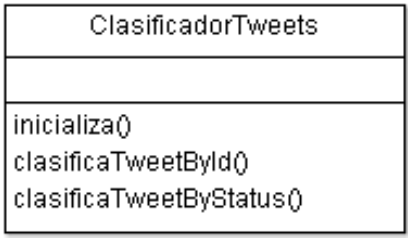
EntrenamientoTweets

EntrenamientoTweets se encarga de lanzar los entrenamientos personalizados para nuestra aplicación, esto es, a partir de una lista de entrenamiento de la base de datos genera el fichero arff y a continuación lanza en un hilo el entrenamiento de la red neuronal anteriormente mencionada. También se encarga de guardar en la base de datos el porcentaje de fallo del entrenamiento y en disco el modelo de la red neuronal.



ClasificadorTweets

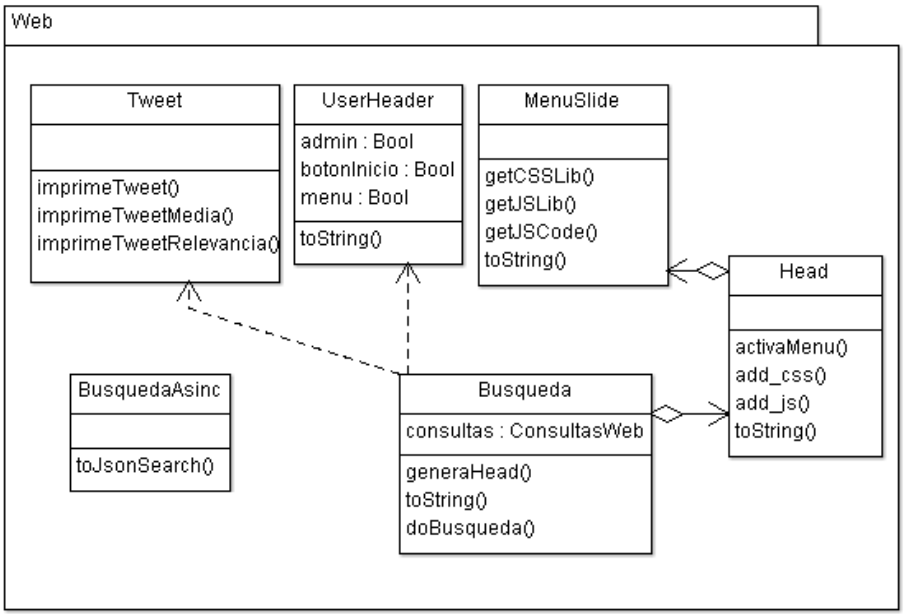
ClasificadorTweets es la clase encargada de clasificar tweets que no son proporcionados por el entrenamiento, esto es, los que llegan según se realiza una búsqueda, también guardar la clase más probable del tweet en la base de datos, la clase pueden ser relevante o no relevante.



3.3.5. Interfaz gráfica, Web

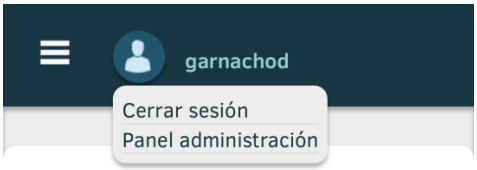
El módulo Web es el encargado de mantener las clases que contienen la interfaz web, desde estas clases se proporciona la entrada de datos a la aplicación y la salida a los usuarios, se ha usado Flask como servidor Web, la aplicación tiene dos tipos de usuarios registrados, administrador y usuarios sin privilegios de administración.

En el módulo Web se usan una serie de clases genéricas que proporcionan una interfaz unificada por la aplicación y evitan duplicar código. En el siguiente esquema solo se representan las clases más significativas.



UserHeader

UserHeader proporciona una interfaz unificada de la aplicación desde esta cabecera podremos desplegar el menú principal de la aplicación y el secundario, desde este último podremos ir al panel de administración si el usuario es administrador.



MenuSlide

MenuSlide es la clase que nos proporciona la interfaz y la usabilidad del menú principal de la aplicación, el estilo de este menú es idéntico al de una aplicación móvil, lo que ha permitido que la aplicación se adapte a estos dispositivos, en la siguiente imagen podremos ver las ventanas accesibles desde este menú.

Head

La clase Head es muy útil para evitar duplicar código HTML, ha sido creada para generar la etiqueta <head> y todo su contenido, pudiendo añadir dinámicamente librerías JavaScript y estilos CSS.

Tweet

Tweet es la clase Python que se encarga de generar el HTML en la parte del servidor, en la parte cliente se ha creado un JavaScript para poder descargar Tweets de manera asíncrona sin hacer esperar al usuario desde el servidor y que genera el mismo estilo.

BusquedaAsinc

BusquedaAsinc en contraposición de las clases mencionadas anteriormente no produce código HTML y como otras clases de la aplicación devuelve al cliente código JSON para conseguir comunicaciones asíncronas desde JavaScript y que mejore la experiencia de usuario, evitando tiempos de espera.

Busqueda

Usaremos la clase Busqueda para explicar el funcionamiento de las interfaces creadas, las interfaces dinámicas de la aplicación realizan consultas a la base de datos, en el caso de la búsqueda, se trata de encontrar tweet ya descargados que coincidan con la consulta y a continuación devolver el código HTML para poder ser visualizado desde un navegador.

4. Entorno de producción e instalación

En esta sección se enunciarán los pasos para poder utilizar la aplicación, el entorno de producción y pruebas utilizado. Encontraremos dos instalaciones totalmente diferenciadas, la aplicación web y el demonio de tareas, estos dos módulos se sincronizan usando la base de datos.

4.1.Sistema operativo

Este proyecto ha sido realizado para sistemas Linux, aunque al haber sido desarrollado en Python no debería suponer mayor problemas en migrarlo a un entorno Windows. Concretamente, el entorno de desarrollo y pruebas ha sido Ubuntu Desktop 14.04 (LTS), el entorno de producción dado por el tutor es Ubuntu Server 12.04 (LTS).

4.1.1. Dependencias software

Aplicaciones externas

- Base de datos **PostgreSql** 9.X
- **Apache Server** con mod_wsgi (8), este punto es necesario se la aplicación ha de arrancar con el inicio del sistema operativo, entorno de producción.
- **Weka,** Aplicación de aprendizaje automático, sólo usada como test del software

Librerías Python 2.7.X

- Servidor web **Flask**
- **Psycopg** conexión con la base de datos PostgreSql
- **Python Daemon** para la creación del demonio en segundo plano
- **Twython** librería de conexión con Twitter

Librerías JavaScript

Las librerías JavaScript no son instalables y se proporcionan con el proyecto.

- **Chart.js** visualización de gráficas
- **Progressbar.js** barra de progreso
- **JQuery.js** librería JavaScript que simplifica la implementación en este lenguaje
- **JQuery mmenu** menú con estilo aplicación móvil

4.2.Instalación del demonio de tareas

Para la creación de demonios en Python es necesario usar la librería **Python Daemon** u otras, esta librería nos proporciona una ejecución continua, esto es, ejecuta el método run() hasta que se para de manera externa, útil si se desea hacer un servidor. Este demonio no se arranca cuando arranca el sistema operativo, por lo que se ha optado por hacer un programa residente Linux. Para ello es necesario crear un programa bash e insertarlo en el arranque. En la siguiente figura se muestra el código desarrollado.

```
#!/bin/bash

# /etc/init.d/demonioTareasTwitter
# Provides: demonioTareasTwitter
# Required-Start: $all
# Should-Start: $all
# Required-Stop:
# Should-Stop:
# Default-Start:  2 3 4 5
# Default-Stop:   0 1 6
# Short-Description: Test daemon process
# Description:    Runs up the test daemon process

# path to app

APP_PATH=

case "$1" in
  start)
    echo "Starting server"
    python $APP_PATH start
    ;;
  stop)
    echo "Stopping server"
    python $APP_PATH stop
    ;;
  restart)
    echo "Restarting server"
    python $APP_PATH restart
    ;;
  *)
    # Refuse to do other stuff
    echo "Usage: /etc/init.d/demonioTareasTwitter.sh
{start|stop|restart}"
    exit 1
    ;;
esac
exit 0
```

En la figura podemos ver la estructura de un programa residente Linux típico con el módulo usado, dado que la clase Python utiliza una base de datos, es necesario que se arranque después de este programa.

Debemos crear la ruta del Log.

```
mkdir -p /var/log/demonioTareasTwitter
```

Copiar el script a /etc/init.d, añadir permisos de ejecución y transferir la pertenencia a root.

```
sudo chown root:root /etc/init.d/demonioTareasTwitter.sh
sudo chmod u+x /etc/init.d/demonioTareasTwitter.sh
```

Por último se habilita el script para la ejecución automática:

```
sudo update-rc.d demonioTareasTwitter.sh defaults
sudo update-rc.d demonioTareasTwitter.sh enable
```

5. Pruebas

En las siguientes pruebas se detallarán los procesos realizados para comprobar el correcto funcionamiento de la aplicación, analizar el rendimiento y la integración de todos los módulos.

5.1. Rendimiento API Twitter

Recuperación de Tweets de un usuario

En esta prueba se va a comprobar la velocidad de recuperación de tweets dado un usuario, en una primera versión recuperaremos 100 elementos y los guardaremos en la base de datos, en una segunda versión recuperaremos todos los datos posibles, 3200 tweets es el límite de la API, pero al haber retweets nos dan ambas búsquedas 3400 tweets.

Versión	Tiempo Base Datos	Tiempo API	Tiempo Total
1	61.21	16.65	78.13
2	59.87	18.38	78.79

Aunque ambas búsquedas den resultados similares, se ha usado la versión 1, porque así podemos dar resultados al usuario antes de terminar la búsqueda completa.
Dada estas descargas, podemos decir que se recuperan unos 200 tweets por segundo.

Recuperación de Tweets dada una palabra o palabras claves

Con esta prueba tratamos de obtener el rendimiento dadas diferentes consultas a Twitter, estas consultas son palabras concretas o hastags, para ver si Twitter hace diferenciación de índices, se ha impuesto un límite de descarga de 2000 tweets.

Versión	Tiempo Base Datos	Tiempo API	Tiempo Total
#hastag	46.65	15.17	61.94
#hastag #hastag	22.04	17.68	40.22
1 palabra	19.4	12.76	32.33
2 palabras	19.24	14.93	34.36
4 palabras	18.6	15.63	34.53

Se puede comprobar que el tiempo relativo en la API al conseguir tweets dado un hastag es 24% del tiempo, que es menor que el tiempo relativo de conseguir tweets dada una palabra 40%, también si relacionamos la prueba anterior con esta, podemos ver como el tiempo relativo de la API aumenta y el de la base de datos disminuye.

5.2. Rendimiento de la base de datos

Ya hemos comprobado el rendimiento de la base de datos en la inserción de tweets en el anterior apartado, ahora comprobaremos las búsquedas y para ello usaremos la búsqueda más compleja de la aplicación que es la de buscar tweets por el contenido del mismo.

Recuperación de Tweets desde la base de datos

Con esta prueba tratamos de evaluar la base de datos desde la recuperación de información, esto es, si es escalable este sistema de búsqueda.

Base de datos con 29K tweets:

Tiempo en recuperar tweets con una palabra muy común 490ms – 2000 tweets

Tiempo en recuperar tweets con una palabra poco común 72ms – 196 tweets

Base de datos con 350K tweets:

Tiempo en recuperar tweets con una palabra muy común 720ms – 2000 tweets

Tiempo en recuperar tweets con una palabra poco común 283ms – 550 tweets

Podemos ver como el tiempo ha aumentado, también podemos ver como el principal problema es más la cantidad de datos retornados que la consulta en la base de datos, por estos motivos supondremos que la base de datos soportará la carga en un sistema en producción, es posible que en con millones de tweets haya que distribuir la base de datos o pensar en otro paradigma.

5.3. Algoritmos de clasificación automática

Esta prueba trata de evaluar los algoritmos de clasificación implementados y así decidir cuál debemos usar, usaremos un 80% entrenamiento 20% test.

Algoritmo	Nº de patrones	Implementación propia	Weka
Naïve Bayes	121	26.08%	25%
Perceptrón multicapa	121	13.04%	12.5%
Naïve Bayes	42	28.57%	25%
Perceptrón multicapa	42	42.85%	37.5%
SVM	121		12.5%
KNN	121		16.6%
Boosting	121		25%

Dados los resultados podemos decir que el perceptrón multicapa es la mejor opción si tenemos un conjunto de entrenamiento grande del que puede inferir con mayor capacidad, por esto nos hemos esforzado en realizar una interfaz de entrenamiento rápida y sencilla. Las redes neuronales tienen un algoritmo de aprendizaje más lento que otros clasificadores, también este clasificador una vez entrenado es uno de los algoritmos más rápidos en clasificación, y eso nos interesa cuando tengamos que procesar un gran número de tweets.

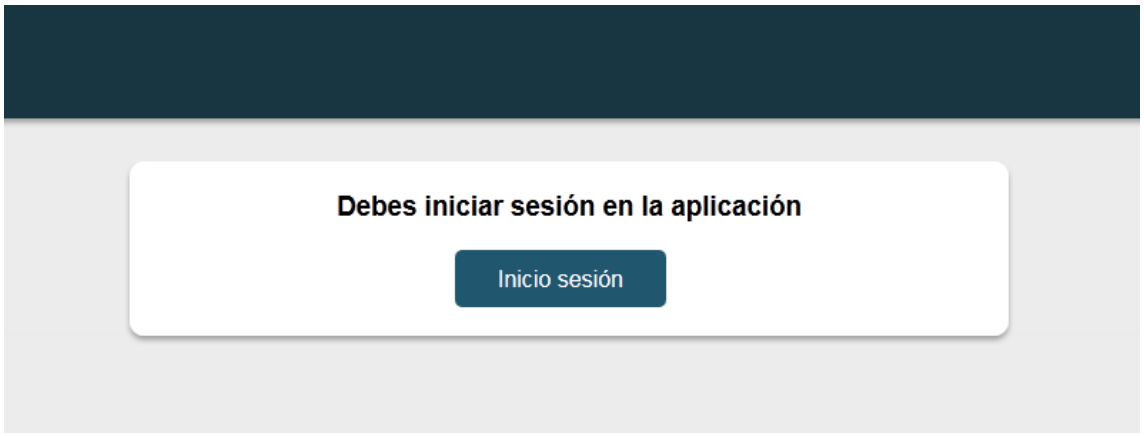
También se ha comentado que en posteriores iteraciones del proyecto se podría migrar la clasificación a un programa externo o librería, dado el diseño no debería suponer mayor problema usar Weka.

5.4. Pruebas de la interfaz Web e integración

En las siguientes pruebas se validará la interfaz web y a la vez se realizarán las pruebas de integración en este punto dado que es el módulo superior e interactúa con los demás, estas pruebas se han realizado con Firefox y Google Chrome.

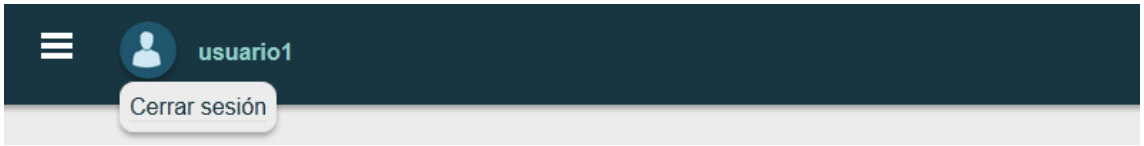
Inicio de sesión, privilegios de usuarios

Todas las páginas comprueban que el usuario haya iniciado sesión por seguridad y si no se ha iniciado sesión muestra una página que redirecciona al inicio de sesión.



Para el inicio de sesión se necesita conocer nombre de usuario y contraseña, una vez introducidos los datos correctamente se puede ver la página principal, todas las páginas controlan los permisos de usuario y si es un administrador se muestra la ruta al panel de administración.

En cambio un usuario no administrador, no puede ver el link, ni acceder aunque conozca la ruta.



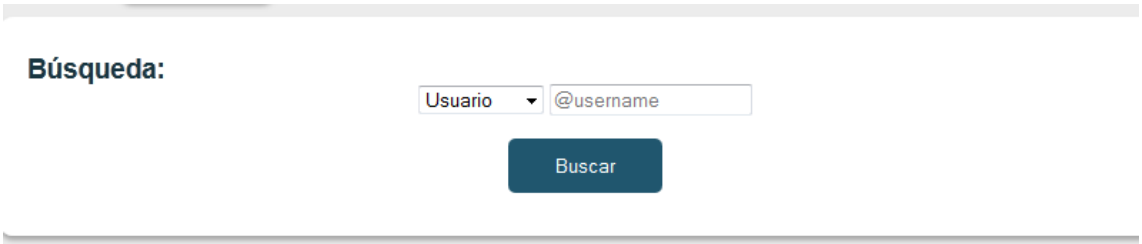
El tiempo de carga de estas páginas es de 0.19 segundos en Firefox y 0.083 segundos en Google Chrome, estando los dispositivos en la misma subred.

El panel de administración contiene dos formularios, uno para crear usuarios de la aplicación y otro para añadir claves de la API de Twitter, como se ha mencionado anteriormente, sólo puede acceder un administrador.

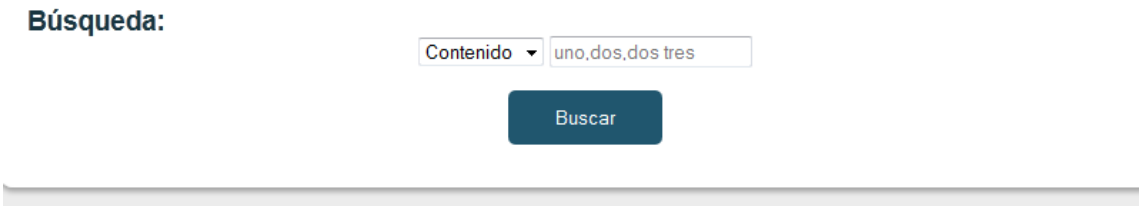
Búsquedas internas y en Twitter

La aplicación es capaz de realizar dos tipos de búsquedas bien diferenciadas, tweets de un usuario determinado, y del contenido del tweet, estas búsquedas se realizan simultáneamente en Twitter y en la base de datos.

Búsqueda de los tweets de un usuario

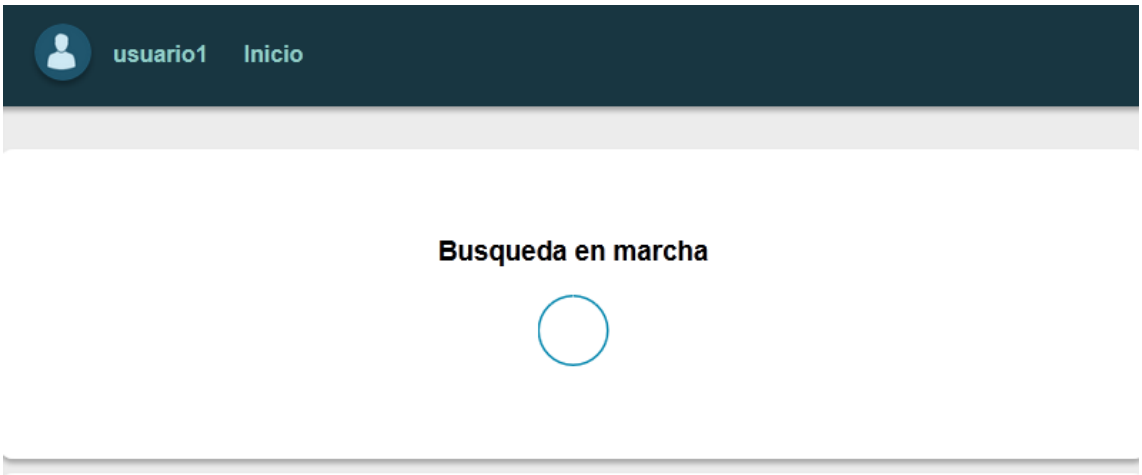


Búsqueda de los tweets por su contenido



Una vez realizada la búsqueda, la aplicación nos devolverá los tweets que coincidan con los parámetros de entrada que estén en la base de datos y lanzará un hilo Python para descargar contenido de Twitter, otro hilo JavaScript se encargará de recuperar el contenido nuevo que se está descargando, estas pruebas han supuesto un trabajo de integración muy grande al tener que sincronizar los proceso en servidor y cliente.

En esta primera imagen podemos ver como se recuperan datos de la base de datos para consulta “3D”



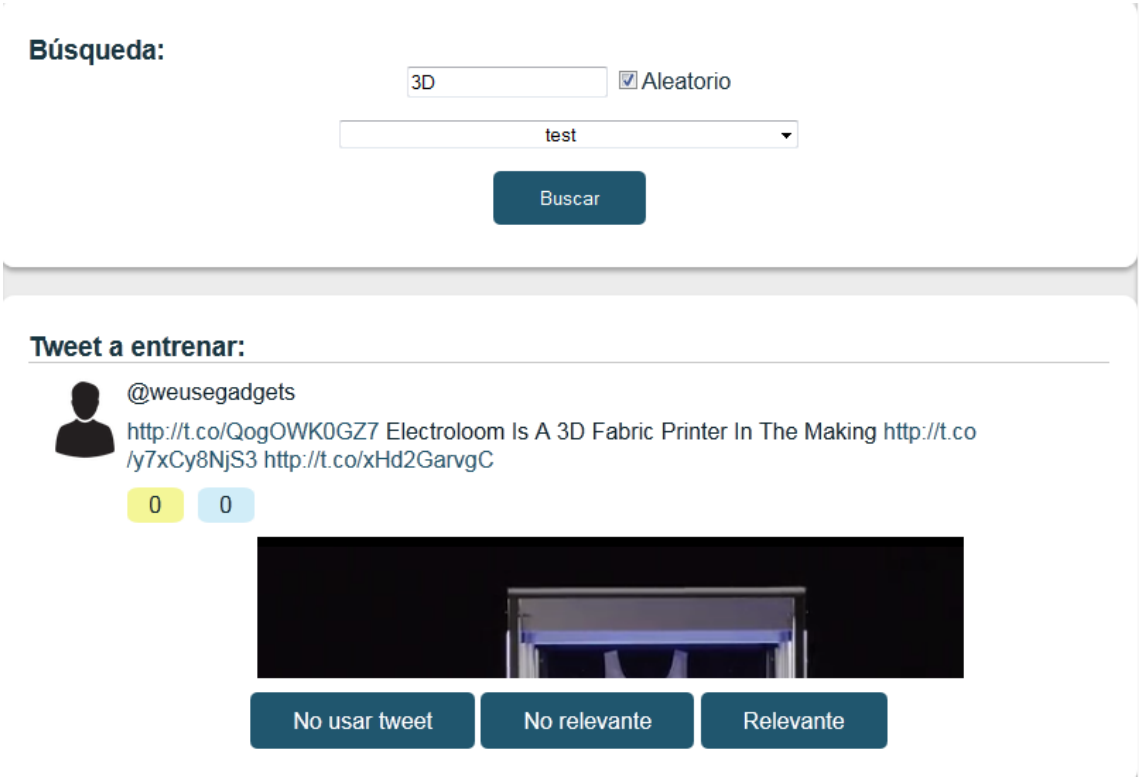
En la segunda imagen podemos ver como ya se han recuperados nuevos tweets de la red social y el hilo JavaScript ha insertado el contenido en la web, búsqueda continua hasta que no queden más tweets que no estén en la base de datos o cuando llegue el límite de la API.

El tiempo de carga de estas páginas varía según los tweets recuperados, varía desde 0.3 segundos si se recuperan desde la base de datos unos 100 tweets hasta los 0.9 segundos si se recuperan 2000 tweets. Este tiempo es el necesario para cargar el DOM de la Web, no se incluye el tiempo de carga de imágenes externas.
También se puede ver en estas pruebas que la recuperación en la base de datos es limitada y puede no dar resultados satisfactorios.

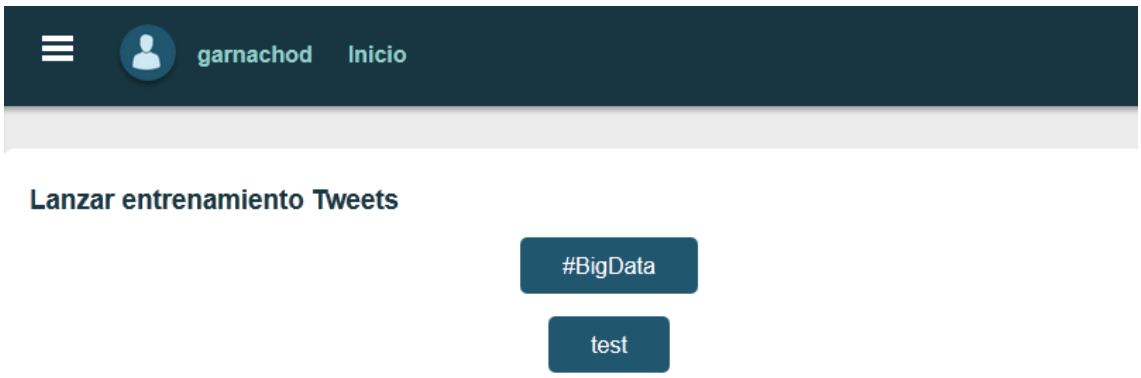
Entrenar usando tweets

En esta prueba vamos a comprobar el correcto funcionamiento del módulo de entrenamiento de la web, así como la integración con el entrenamiento y clasificación de la red neuronal.

Para el entrenamiento web se ha creado un hilo JavaScript que se encarga de enviar información al servidor y recibir nuevos tweets de entrenamiento cada vez que se vote, esto evita tiempos de carga no necesarios. El usuario es el encargado de solicitar la información que pueda ajustarse con la búsqueda y a la vez es el encargado de comunicar a la aplicación si es un tweet relevante o no relevante, también se proporciona un botón de no usar, por si el tweet no contiene información útil, por ejemplo, es sólo una imagen. El usuario también debe elegir a que lista de entrenamiento quiere que vaya el tweet.
Se ha intentado mantener la simplicidad al máximo.



En la siguiente imagen podemos ver como se lanza un entrenamiento por parte del usuario, el usuario debe solicitar el entrenamiento del clasificador cuando haya terminado de generar el conjunto de entrenamiento, porque al ser una tarea que necesita un cálculo intensivo no se puede lanzar cada vez que inserte un tweet en el conjunto de entrenamiento.

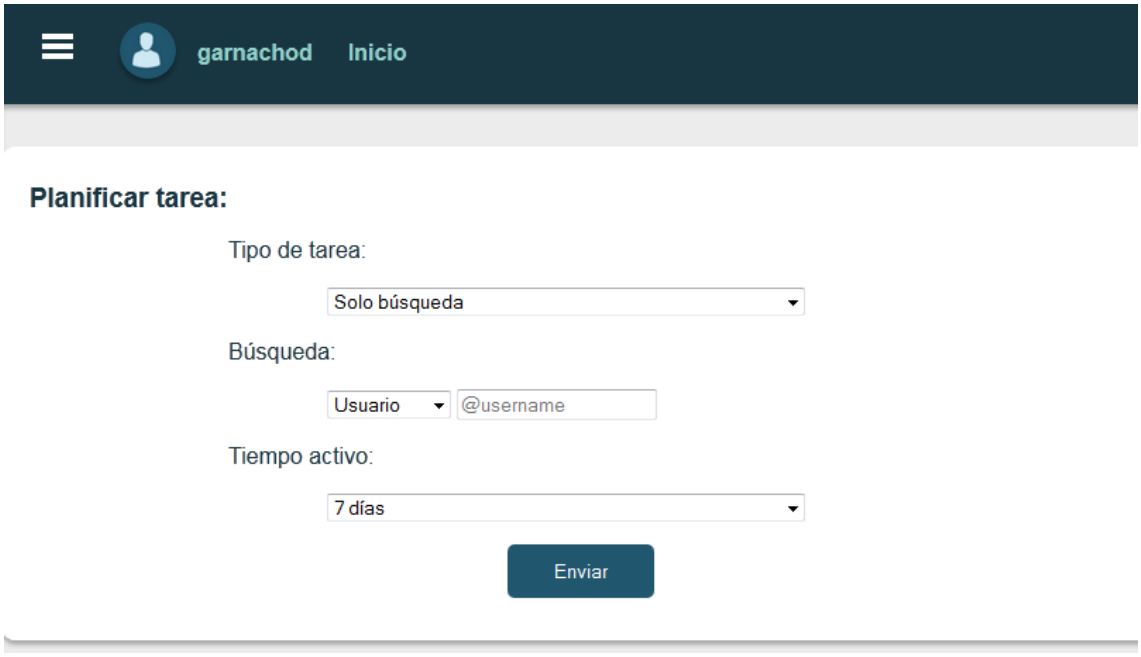


Tareas en segundo plano

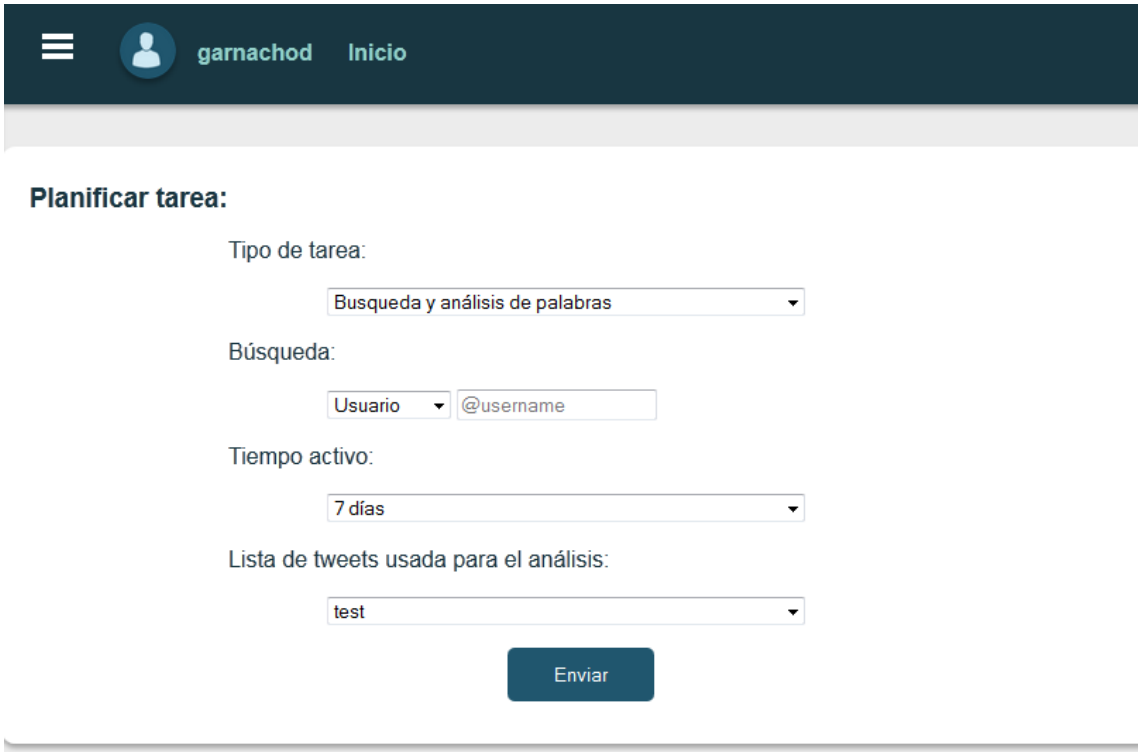
Con las siguientes pruebas comprobaremos el correcto funcionamiento de las tareas en segundo plano, tanto de recuperación como recuperación con clasificación, estas tareas utilizan el módulo MachineLearning, Demonio, SocialAPI y para la visualización el módulo Web.

Planificación de una tarea en segundo plano

En la siguiente imagen vemos cómo crear una tarea en segundo plano de búsqueda solamente, al igual que en la búsqueda principal, se puede elegir buscar a un usuario o por el contenido de los tweets.



En la siguiente imagen podemos ver una tarea con análisis, para ello debemos elegir también a que lista de entrenamiento queremos lanzar los tweets que se recuperen con la búsqueda.

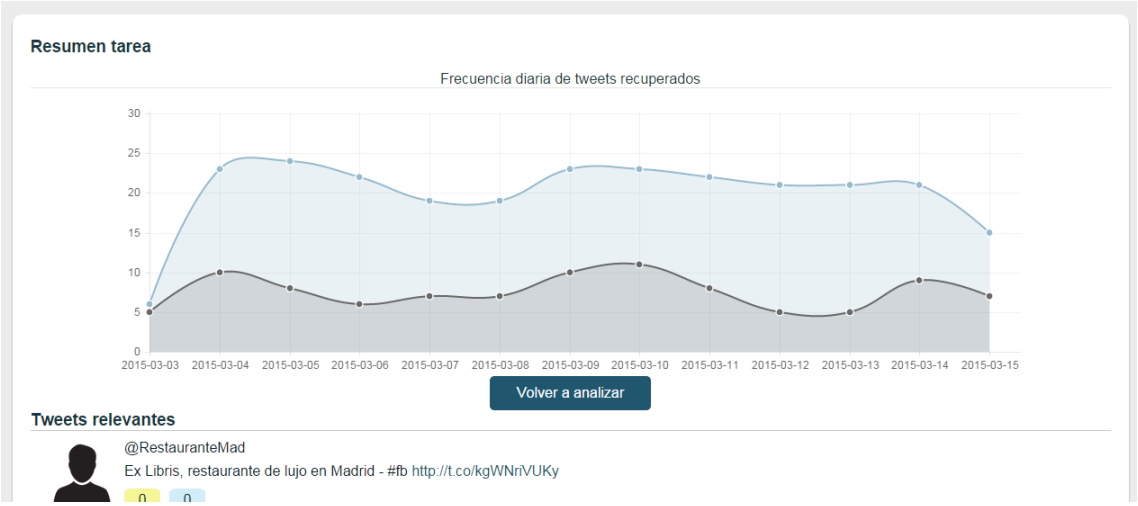


Visualización de una tarea en segundo plano

Las tareas en segundo plano nos ofrecen a parte de la búsqueda continuada en el tiempo, nos permiten generar análisis de tiempos, o por ejemplo si recuperáramos seguidores conseguiríamos más información que la que nos da la API, por ello se ha generado una gráfica dinámica de frecuencias de tweets al día en cada tarea.

En la siguiente imagen podemos ver un resumen de la tarea en segundo plano de búsqueda, muestra todos los tweets recuperados, se puede también ver una curva de frecuencias típica de un Hashtag muy concentrado en el tiempo.

En la siguiente imagen vemos una tarea con análisis, este resumen solo mostrará los tweets clasificados como relevantes y dos curvas de frecuencia, la curva azul representa los recuperados y la gris los clasificados como relevantes. El botón de volver a analizar sirve si se ha vuelto a entrenar la red neuronal, para filtrar de nuevo los tweets con ese entrenamiento.



6. Conclusiones y trabajo futuro

En este trabajo fin de grado se ha diseñado y programado una aplicación web para la búsqueda y el análisis de datos relativos a la red social Twitter.

El módulo de captura de datos ha resultado una pieza clave para funcionamiento global de la aplicación, ha permitido centralizar la búsqueda de Twitter, ofreciendo un diseño flexible por si en un futuro se desea ampliar el diseño a otros datos o redes sociales, si bien ha resultado un quebradero de cabeza la obtención y almacenaje de los datos, se ha conseguido un rendimiento suficiente en búsquedas, si se quisiera mejorar el almacenamiento se podrían realizar inserciones en la base de datos asíncronas, evitando tiempo de espera entre consulta de la API.

El módulo Machine Learning ha resultado un reto al no conocer métodos eficaces para la correcta clasificación de texto, más concretamente, la clasificación de tweets supone una complejidad añadida al contener muy pocas palabras. Se ha visto también que el entrenamiento de una red neuronal con tantas entradas puede llevar al sobreaprendizaje y para ello se han creado un conjunto de validación en entrenamiento.

Se considera un éxito la flexibilidad del módulo Demonio, la posibilidad de generar tareas dinámicamente y con total independencia de la implementación ofrecen la posibilidad de añadir funcionalidad de manera rápida. Este módulo también ofrece al usuario poder monitorizar gran cantidad de tweets sin tener que leerlos en el momento y almacenarlos para un posterior análisis.

La interfaz web proporciona un acceso a los datos simple y se intenta mejorar la experiencia a la ofrecida por twitter, la interfaz se ha creado con la intención de que pueda usar por cualquier usuario, este no necesita conocer los mecanismos internos, solo debe ser capaz de usar su conocimiento para buscar y clasificar tweets, la herramienta le ofrecerá los datos una vez hayan sido procesados, también es útil que desde cualquier dispositivo se pueda acceder a la aplicación, ya sea móvil, tablet o PC.

Es muy importante tener en cuenta que el análisis que se hace de la red social es limitado y no es capaz de dar información de la distribución general de contenido, ni de usuarios, esta aplicación puede suponer la base para un proyecto análisis de redes sociales más profundo, como puede ser la detección de comunidades sociales o análisis de marketing online, usando la clasificación automática para el filtrado de la información relevante.

6.1.Trabajo futuro

Las posibilidades de mejora y ampliación que ofrece este trabajo fin de grado son muchas, se proponen las siguientes ideas:

- **Añadir más redes sociales**. Una red social es capaz de ofrecernos información sesgada de cómo se comportan sus usuarios en ella, por eso sería interesante añadir más información y a la vez intentar comparar las diferentes redes.

- **Análisis de la distribución de la red**. Sería muy interesante conocer cómo se conectan e interactúan los usuarios, cual es la distribución de seguidores, esto puede ayudarnos por ejemplo en marketing a aumentar la reputación de una marca o a la detección de comunidades.

- **Optimizar almacenamiento y búsqueda**. Se ha comentado en otros puntos del proyecto la posibilidad de distribuir los datos en diferentes servidores, así poder almacenar una mayor cantidad de información, esto también supone cambiar el paradigma de búsqueda, teniendo que generar buscadores similares a los usados en búsqueda web.

- **Mejorar la clasificación automática**. La clasificación automática implementada en el proyecto supone una herramienta útil, pero se debería investigar si es escalable o si hay métodos que puedan llegar a dar mejor resultado, como puede usar TF-IDF en vez de la frecuencia o añadir otros campos.

- **Aumentar la capacidad de cómputo**. Sería bueno poder aumentar el número de servidores según la carga de trabajo aumente o por ejemplo distribuir las tareas en segundo plano entre servidores, haciendo la recopilación de datos más rápida. El problema a este punto es la cantidad de claves necesarias para la API de Twitter.

- **Clasificación de sentimientos e imágenes**. Investigar la inclusión de módulos externos de clasificación de sentimientos nos ayudaría a aumentar la información que tenemos de la red social, de un usuario o de un Hashtag. La clasificación de imágenes ayudaría a darnos más información sobre los tweets, se ha estimado que el 25% de los tweets recuperados contienen imágenes, se podrían usar librerías externas como Caffe (13).

7. Referencias

1. **Twitter.** Twitter Analytics. [En línea] Twitter, 2015. [Citado el: 3 de Mayo de 2015.] https://analytics.twitter.com/.

2. **Topsy Labs, Inc.** Twitter Search, Monitoring, & Analytics. *Topsy.* [En línea] Topsy Labs, Inc, 2015. [Citado el: 3 de Mayo de 2015.]

3. **Twitonomy.** Twitter analytics and much more. *Twitonomy.* [En línea] Diginomy Pty Ltd, 2013. [Citado el: 11 de Octubre de 2014.]

4. *The WEKA Data Mining Software: An Update; SIGKDD Explorations.* **Hall, Mark, y otros, y otros.** 1, Vol. 11.

5. *Scikit-learn: Machine Learning in Python.* **Pedregosa, F. and Varoquaux, G. and Gramfort, A. and Michel, V.** 2011, Vol. 12.

6. **Apache Software Foundation.** Scalable machine learning and data mining. *Apache Mahout.* [En línea] [Citado el: 11 de Marzo de 2015.] http://mahout.apache.org/.

7. **The Apache Software Foundation.** The Apache HTTP Server Project. *Apache.* [En línea] 1996. [Citado el: 19 de Noviembre de 2014.] http://httpd.apache.org/.

8. **Dumpleton, Graham.** Apache module that implements a WSGI. *MOD_WSGI.* [En línea] [Citado el: 19 de Noviembre de 2014.] https://github.com/GrahamDumpleton/mod_wsgi.

9. **Django Software Foundation.** The Web framework for perfectionists with deadlines. *Django.* [En línea] Django Software Foundation, 2010. [Citado el: 10 de Octubre de 2014.] https://www.djangoproject.com/.

10. **Ronacher, Armin.** Flask (A Python Microframework). *Flask.* [En línea] Pocoo, 2014. [Citado el: 10 de Octubre de 2014.] http://flask.pocoo.org/.

11. **McGrath, Ryan y Helmick, Mike.** Actively maintained, pure Python wrapper for the Twitter API. Supports both normal and streaming Twitter APIs. *Twython.* [En línea] [Citado el: 25 de Octubre de 2014.] https://twython.readthedocs.org/en/latest/.

12. **Fausett, Laurene.** *Fundamentals of Neural Networks: Architectures, Algorithms and Applications.* New Jersey : Prentice-Hall, 1994. ISBN-13: 9788131700532.

13. **Jia, y otros, y otros.** Caffe deep learning framework. *Caffe: Convolutional Architecture for Fast Feature Embedding.* [En línea] Berkeley, 2014. [Citado el: 19 de Mayo de 2015.] http://caffe.berkeleyvision.org/.